IAPR Workshop on CV - Special Hardware and Industrial Applications OCT.12-14, 1988, Tokyo

IMPLEMENTATION ISSUES ON PARALLEL ALGORITHMS USING PIPELINE ARCHITECTURES IN VISUAL INSPECTION

Amelia Fong

*Department of Computing and Information Science**
University of Guelph
Guelph, Ontario
Canada N1G 2W1

ABSTRACT

Parallel architectures are increasingly being used for vision applications where speed is an important requirement, such as in manufacture visual inspection. There is currently much research activities in image processing algorithms using pipeline architectures for industrial applications. However, for a successful inspection system, various implementation details need to be considered as well. In this paper implementation issues involving refresh memory usage are discussed, and techniques to avoid disk transfer are proposed. Issues on software environment to support rapid program development in order to reduce software cost are also discussed. Our discussion will be illustrated by an example from an algorithm for a non-linear order-statistical filter.

I. Introduction

Due to the amount of data and speed requirement, parallel architectures are increasingly being used for computer vision applications, such as in pattern recognition problems involving image data. These include manufacture visual inspection, robotic vision as well as various forms of medical imaging for diagnostic purposes. In most industrial applications such as in visual inspection there is an important requirement that the cost be kept low so that the manufacture process remains cost effective [Chin, 86], [Uhr, 86]. Recently, several vision systems have been proposed using pipeline architectures [Petkovic, 86], [Persoon, 88]. Pipeline architectures have fast performance and are commercially available, allowing applications to be tested before dedicated equipment need to be built. Currently there are much research interest and publications in parallel algorithms using pipeline architectures, including various geometric features [SHD, 85], [Sanz & Dinstein, 87], polygonal masks [Sanz 88], [SDP, 88], projections [Sanz & Hinkle, 88], local maximum and minimum [Dinstein & Fong , 88], and a class of detail preserving filters [Fong , 88a].

*This research was partially supported by the Natural Science and Engineering Research Council of Canada.

Because of the speed requirements, it is natural to select the algorithm which will do the job and uses the least number of pipeline passes. However there are other considerations which affect the total time taken, such as the amount of refresh memories available on the hardware and how these affect the executions of various algorithms. Care must be taken when implementing these algorithms so that disk transfers are avoided, which are expensive when large amount of data such as in image applications, is involved. With many implementation details the programming effort required can be substantial which will add to the total cost of the inspection system. Hence, these implementation issues are important ones. The paper is organized as follows. Section II describes the pipeline architectures assumed. Section III reviews an example algorithm used for illustration in our discussion. Section IV discusses implementation issues involving refresh memory usage. Section V contains discussion on software environment. Section VI concludes the paper.

II. Pipeline Architectures

In this section, we describe the abstract machine architecture we are assuming. We assume a high speed pipeline processor which takes input from a number of image refresh memories or from a digitizer. The processor typically consists of several parallel stages of processing, such as described below. An example of such an architecture is the DeAnza IP8500, whose pipeline processor can operate ate video refresh rate, i.e. 30 frames/sec. Other typical features include various display capabilities, input/output devices such as joysticks, lightpens and trackballs, programmable cursor units, etc. which are not relevant in our present discussion.

We describe a prototype general purpose pipeline image processor with more details. It typically consists of an input selection stage, a multiplier stage which (optionally) multiply pairs of selected inputs. This is followed by one or more ALU stage(s) which performs arithmetic and/or logical operations. These are followed by a comparator stage, a shifter stage and a function table stage which can be used as a look-up table.

In our abstract machine model the image processor consists of a short pipe of processing units. The processing units are general purpose units such as

ALU's and comparators and shifters. We assume also a feed back loop from the pipeline processor to refresh memories whose number can vary depending on the particular configuration. A schematic diagram of our abstract machine model is shown in Fig. 1. We assume each refresh memory has an on board look-up table for performing the basic transformations such as translations in the x or y or both directions as the data streams from the memory. What we have assumed is a general purpose pipeline architecture so that our discussions on implementation issues are generally applicable.

III. An Example Algorithm

As an example, we shall consider a median filter over all samples of a 3×3 window. This is an adaptation from an algorithm in [Fong, 88c] where the proof of correctness can be found.

We shall use the following notation. Define Tr[A, (p,q)] to be the resulting image of translating the image A p pixels in the x direction, and q pixels in the y direction. When p (or q) is negative, it indicates that the translation is in -x direction (-y direction). In the following pipeline steps, max and min denote the use of the hardware comparator which allowed tow images to be compared pixelwise.

```
Algorithm:
Let P be the input image
Let $P_1=Tr[P,(1,0)]$
Let $P_2=Tr[P,(2,0)]$
The following pipeline passes are performed:
    1.  $\max(P,P_1)=X_1$
    2.  $\min(P,P_1)=X_2$
    3.  $\max(X_2,P_2)=X_3$
    4.  $\min(X_1,X_3)=X_4$
    5.  $\max(X_1,P_2)=X_5$
    6.  $\min(X_2,P_2)=X_6$
Let $Y_1=Tr[X_6,(0,1)]$
    $Y_2=Tr[X_6,(0,2)]$
Perform the following pipeline passes
    7.  $\max(X_6,Y_1)=X_7$
    8.  $\max(X_7,Y_2)=X_8$
Let $Z_1=Tr[X_5,(0,1)]$
    $Z_2=Tr[X_5,(0,2)]$
Perform the following passes
    9.  $\min(X_5,Z_1)=X_9$
    10. $\min(X_9,Z_2)=X_{10}$
Let $Q_1=Tr[X_4,(0,1)]$
    $Q_2=Tr[X_4,(0,2)]$
Perform the following passes
    11. $\max(X_4,Q_1)=X_{11}$
    12. $\min(X_4,Q_1)=X_{12}$
    13. $\max(X_{12},Q_2)=X_{13}$
    14. $\min(X_{11},X_{13})=X_{14}$
    15. $\max(X_8,X_{10})=X_{15}$
    16. $\min(X_8,X_{10})=X_{16}$
    17. $\max(X_{16},X_{14})=X_{17}$
    18. $\min(X_{15},X_{17})=X_{18}$
Then $X_{18}$ is the output image.
```

We shall use this algorithm as an example in the discussions in the following sections.

IV. Implementation Issues on Memory Usage

In this section, we discuss various implementation issues involving the use of refresh memories, a valuable resource. In many algorithms using pipeline architectures, more than one pipeline pass are required. Hence intermediate results are stored in the refresh memories whose content are routed back to the pipeline processor for further transformation. Often, enough refresh memories are assumed so that processing can continue until completion. In reality, depending on the particular configuration, one might not have enough refresh memories to hold all the intermediate results. This is especially true if many different processing steps are needed in any one inspection application, such as smoothing followed by edge detection and other segmentation steps. In such cases, writing to disk and reloading from disks may be necessary. Due to the amount of data involved with images, this data transfer can significantly affect the overall performance.

There are various techniques pertaining to implementation which can minimize the number of refresh memories required. We shall discuss them below.

4.1 Multiple output copies

This technique can be illustrated by our example above. Consider steps 12 and 13 in the above algorithm. Images Q_1 and Q_2 are used as operands. Note that Q_2 is Tr[Q_1, (0,1)] based on the relationships of Q_1 and Q_2 to X_4. Hence when X_4 is computed, the output can be routed to two separate refresh memories. By setting the on-board registers on the two refresh memories, both Q_1 and Q_2 can be obtained by translating the input as it streams out of the respective memory. This saves extra pipeline passes for setting up operands.

4.2 Original image recovery

This can be illustrated by the algorithm above. The algorithm involves manipulating the images P, the original image, P_1 = Tr[P, (1,0)] and P_2 = Tr[P, (2,0)]. It can be seen that at any given time only one of these images need to be around. P can be recovered from either P_1 or P_2 by translating in the opposite direction so that it need not be saved nor reloaded. By careful use of this technique, we find that the above algorithm requires only 6 refresh memories, assuming the use of one comparator at each pipeline step.

4.3 Loading of input images

Sequence of input images should be loaded in succession as required, to avoid occupying refresh memories before it is being used. Due to space limitation, other algorithms which use more than one input image are not included here. An example can be found in the algorithms for hybrid filters in [Fong, 88a].

4.4 Multiple processing units

Multiple processing units allow more parallelism and usually improve the time required considerably. For example, the above algorithm can be implemented using 10 pipeline passes, assuming two comparators which can be set independently, as is the case in the DeAnza IP8500. However, this results in more intermediate results being simultaneously generated which need to be stored in refresh memories. In our example algorithm, if two comparators are used in parallel, 9 refresh memories are required. Hence, in the case where a configuration does not have enough refresh memories or where the memories need to be saved for other usage, an algorithm using less than all processing units can be used in order to avoid disk transfer, which may be more costly.

4.5 Use of copy instead of disk transfer

Sometimes, an extra pipeline pass may be used to copy an operand instead of saving the image and reloading it from disk. This can reduce the number of refresh memories required at the expence of one extra pass. In the above algorithm, this technique can be used to reduce the number of refresh memories required from 9 to 8. However, in many instances, the steps need to be reorganized to make this possible.

V. Implementation issues on software environment

As discussed earlier, given a particular algorithm for a pipeline architecture, a number of implementation details need to be designed, in order to take full advantage of the hardware. This often requires resequencing of the steps, with an eye towards increase parallelism, while at the same time conserving refresh memories used, avoiding disk transfer whenever possible. Once these details are worked out, each pipeline pass must be programmed explicitly, including specifying all bus connections to and from individual memory and setting up all on-board look-up tables. As a result, the programs are difficult to develop and maintain.

The amount of time needed to develop and debug the programs can greatly undermine the cost-effectiveness of a pipeline system used for inspection. In the following, we briefly discuss the software environment that can enhance ease of implementation.

5.1 High level language construct

Implementation effort can be greatly reduced if a high level language environment that capture both the capabilities of the hardware as well as the domain of applications is available. It should provide high level data structures as well as abstract operations and transformations appropriate to the image processing domain. A preprocessor need to be developed to parse and generate good low level code for the hardware. Debugging facilities should be provided as well. One such proposal can be found in [Fong, 88b].

5.2 Software design tools

Given the design strategies already discovered on the implementation details of algorithms on pipeline architectures, it would be useful to have design tools that facilitate or automate some or all of these steps. A design toolkit may propose alternate sequencing of the pipeline steps. Transformations based on algebraic properties may be applied to uncover opportunities for optimization. A software tool for automatic assignment of refresh memories and for setting up the memory registers may be provided by the toolkit as well.

5.3 Graphical interface

For the experienced user who knows the hardware intimately, a graphical interface which allows routing through the pipeline processor by indicating it on the screen would be a useful tool. This is invaluable in the case where maximum performance is required in specific area of an application and rapid low level coding may be achieved by explicit routing.

VI. Conclusion

In this paper, we have discussed various issues pertaining to the implementation of parallel algorithms on pipeline architectures. In particular, we discussed the implication of refresh memory usage and propose techniques useful for detail design and sequencing of pipeline steps, We also discussed the importance of program development environment in order to reduce software cost and improve overall performance of an inspection system.

References

[Chin, 86] R.T. Chin, "Algorithms and Techniques for Automated Visual Inspection", in Handbook of Pattern Recognition and Image Processing, ed. by T.Y. Young and K.S. Fu, Academic Press, 1986, pp. 587-612.

[Dinstein and Fong, 88] I. Dinstein and A.C. Fong (Lochovsky), "Computing Local Minima and Maxima of Digital Images in Pipeline Image processing Systems equipped with Hardware Comparators:, Proceedings of the IEEE, to appear.

[Persoon, 88] E. Persoon, "A Pipelined Image Analysis System Using Customed Circuits", IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-10, Jan., 1988, pp.110-117. pp. 74-90.

[Petkovic, 86] D. Petkovic *et al*, "An experimental system for disc heads inspection", Proc. of the 8th Int. Conf. on Pattern Recognition, Oct. 1986, Paris, France.

[Sanz, 88] J.L.C. Sanz, "A new method for computing polygonal marks in image processing pipeline architectures", Pattern Recognition, to appear.

[Sanz & Dinstein, 87] J. L.C. Sanz and I. Dinstein, "Projection- based Geometrical Feature Extraction for Computer Vision Algorithms: Algorithms in Pipeline Architectures", In IEEE Transaction on Pattern Analysis and Machine Intelligence, Jan. 1987, pp. 160-168.

[SHD, 85] J.L.C. Sanz, E. Hinkle, and I. Dinstein, "A new approach to computing geometric features of digital objects for machine vision, image processing and image analysis: Algorithms in pipeline architectures", in CVPR 85, San Francisco, CA, June 1985.

[SDP, 88] J.L.C. Sanz, I. Dinstein and D. Petkovic, "A new procedure for computing multi-colored polygonal masks in image processing pipeline architectures and its application to automatic visual inspection, "to appear, CACM.

[Sanz & Hinkle, 88] J.L. Sanz and E. Hinkle, "Computing projections in pipeline architectures", IEEE Trans. Acoust., Speech, Signal Processing, to be published.

[Uhr, 86] L. Uhr, "Parallel Architectures for image processing, computer vision, and pattern perception, in Handbook of Pattern Recognition and Image Processing, ed. Young and K.S. Fu, Academic Press, 1986.

[Fong, 88a] Fong (Lochovsky), A.," Computation of a class of detail preserving filters in Pipeline Architectures", to appear in VISION '88, sponsored by Canadian Assoc. of Pattern Recognition, Edmonton, June, 1988.

[Fong, 88b] Fong (Lochovsky), A., "Increasing Software Productivity of Pipeline Image Processing Architectures in Manufacture Inspection Applications", IEEE 1988 International Conference on Systems, Man & Cybernetics, Beijing, China, Aug. 1988, pp.1138-1143.

[Fong, 88c] Fong (Lochovsky), A., "Parallel computation of several classes of median filters using pipeline architectures", Technical Report CIS88D5, Dept. of Computing & Information Science, University of Guelph, June, 1988.

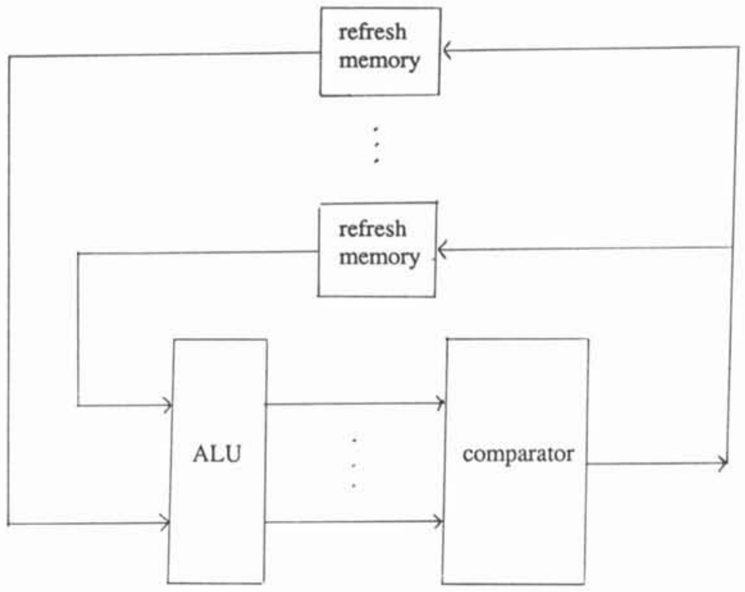

Fig. 1. Schematic Diagram for Pipeline Architectures.